UNIVERSAL PERIPHERAL DEVICE CONTROLLER

Field of the Invention

The present invention relates to sending information from a portable device to a remotely-located computer. More specifically, it relates to data transmission from a computer peripheral device over the Internet to a web server.

Background

Most device communication systems require that custom software, drivers, and/or user interfacing software be installed on a personal computer (PC) in order to allow a peripheral device, such as a Palm Pilot, digital pad, or other peripheral device to communicate with the PC. In order for the peripheral device to transfer data to a remotely-located computer, such as a server, the peripheral device must first transfer the data to a local PC or other computer that has had the required custom software, drivers, and/or user interfacing software installed.

The necessity of installing customized software, drivers and user interface software onto a PC to enable a peripheral device to communicate with that PC or remotely-located PCs or servers creates problems. Excess memory is used, interference with other software can occur, upgrades need to be installed on the PC, the software has to be maintained, conflicts between drivers can develop, as well as conflicts between COM ports and other conflicts between the software required to run the peripheral device and software that runs on the PC for other purposes.

Moreover, whenever a user desires to transfer information from a peripheral device to a remotely-located computer, the user must either find a PC that has had the appropriate software installed or bring the software along so that he can install it on the nearest PC. What is needed is a method and system for transferring data from a peripheral device to a remote computer that is independent of what software is installed on a local PC or other device used to link the peripheral device to the Internet.

Summary

A preferred embodiment of the present invention comprises a software plug-in that allows a peripheral device to communicate via a host (e.g., a personal computer) with a remote server through a communications port. The plug-in is preferably downloaded from the remote server and allows the peripheral device to communicate with that server or other remote servers, and the remote servers are able to communicate and control the peripheral device without any additional software being installed to the host.

Brief Description of the Drawings

FIG. 1 depicts two potential configurations of a preferred system.

FIG. 2 is a flowchart showing preferred functionality of software of a preferred embodiment of the present invention.

Detailed Description of Preferred Embodiments

FIG. 1 depicts two potential configurations of a preferred system. In a preferred embodiment, a user attaches a peripheral device, such as a digital camera **50** or a digitizer pad **60**, to a communications port (whether on a PC **10**, a Web Phone, an Internet-enabled Palm Pilot **30** or another Internet access device) and then uses a web browser to access a system web server **40**.

Upon connecting to the system web server **40**, the user downloads a plug-in to the PC **10**, for example, that allows the peripheral device to communicate to the remote server **40** through the communications port. Herein, the terms "plug-in" and "plug-in computer program" include software such as a browser plug-in, a PRC (also known as a "Palm Resource" or "Palm Application"), or an ActiveX Control.

The plug-in allows the peripheral device to communicate with remote servers of the system and the remote servers are able to communicate with and control the peripheral device. Source code for a browser plug-in written in the C++ programming language and that uses the Netscape Plug-in Application Programming Interface (API) for running on Windows platforms is included in the Appendix at the end of this description.

FIG. 2 is a flowchart showing preferred functionality of the plug-in and steps of a preferred method. A *host* is a device (PC with browser **10**, Internet-enabled Palm device **30**, or other Internet-enabled device) that an *input device* (peripheral device – e.g., digitizer pad **60**, digital camera **50**, non-Internet-enabled Palm Pilot) is connected to via a communications port of the host. As used herein, the term "communications port" includes an RS-232 serial port, a USB port, an infrared port, or a Bluetooth port. Thus, the term "input device" does not include a keyboard or a mouse. In the following description, the actions of the host are controlled by a plug-in that has preferably been downloaded over the Internet. At step **105** a host watches for data from an input device. At step **110** the host checks whether a request from the input device to upload data has been detected. If not, the host continues at step **105** to watch for data from the input device.

If at step **110** a request from the input device to upload data has been detected, then at step **115** the host initiates an upload process, and at step **120** data is transferred from the input device to the host's data storage. The data transfer is performed using the input device's specific communications protocol. This protocol is utilized by the plug-in. In a preferred embodiment, a different plug-in is used for each different communications protocol. In an alternate embodiment, a single plug-in comprises software to enable communications with a plurality of devices that use a plurality of different communications protocols.

At step **125**, the host checks whether the data transfer is complete. If not, then step **120** is repeated and/or continued, as appropriate. If at step **125** data transfer is complete, then at step **130** the host prepares the received and stored data for transmission to a system web server **40**. The data may be reformatted at this step. Preferably, it is packaged into a standard HTML POST command data packet.

At step **135**, the host initiates transmission of the received and stored data to a system web server **40**. At step **140** the data is transferred from the host to the web server **40** through a browser installed on the host and the web server **40**. The data is transferred to the system web server **40** using an API provided by the browser.

At step **145** the host checks whether the data transfer to the web server **40** is complete. If not, then step **140** is continued or repeated, as appropriate. If at step **145** the data transfer is complete, then at step **150** the host reports the status of the data transmission to the user (success or failure). At step **155** the host returns to a monitoring state and repeats step **105**.

Although the present invention has been described with respect to input devices such as digitizer pads and digital cameras, and Internet-enabled devices such as PCs with browsers and Internet-enabled Palm Pilots or other personal digital assistants (PDAs), those skilled in the art will recognize that the invention may be used to transmit data from any input device to a web server, if the input device is configured to transmit data to a PC or other device that can be connected to the Internet.

Appendix

Source code for a browser plug-in written in the C++ programming language and that uses the Netscape Plug-in Application Programming Interface (API) for running on Windows platforms:

```
//=================================================================
#include <stdio.h>
#include <string.h>
#include "npapi.h"
#include <windows.h>
#include "resource.h"

#pragma comment(lib, "Wsock32.lib")

#import "C:\dev\vc\timbrel_plugin\Windows\InkXfer.tlb"
using namespace INKXFERLib;

LRESULT CALLBACK PluginWindowProc( HWND hWnd, UINT Msg, WPARAM wParam,
LPARAM lParam);
const char* gInstanceLookupString = "instance->pdata";

HANDLE      hComm;
int         gConnected = 0;
static unsigned char *inBuffer=NULL;
static unsigned char *outBuffer=NULL;
DWORD       inBufferSize;
DWORD       outBufferSize;

#define kMAX_STRS   25
```

```
char     gMessageTextArray[kMAX_STRS][256]; // = {"Line 1","Line 2","Line
3","Line 4","Line 5","Line 6","Line 7","Line 8","Line 9"};
int               gMessageTextIndex = 0;
int               gNumLines=kMAX_STRS;

typedef struct _PluginInstance
{
         NPWindow*     fWindow;
         uint16        fMode;

         HWND                 fhWnd;
         WNDPROC              fDefaultWindowProc;
         NPP                  gInstance;

         char          gHostName[256];
         char          gHostPort[8];
         char          gUID[8];
         char          gProxyName[256];
         char          gProxyPort[8];
         char          gComPort[8];
         char          gComSpeed[10];
         char          gSourceURL[256];
         BOOL          gVerbose;
         char          gVersion[6];

         BOOL          bTransNote;
         BOOL          gReading;
         DWORD                dwInBufferCount;
         DWORD                dwInBufferIndex;
         DWORD                dwOutBufferCount;
         DWORD                dwFrame;
         DWORD                dwSubFrame;
         DWORD                dwFrameLength;
         DWORD                dwDataLength;
```

```
        DWORD              dwBlockNumber;
        DWORD              dwPreviousBlockNumber;
        DWORD              dwBlockNumberC;
        DWORD              dwBlockStart;
        DWORD              dwNackCount;
} PluginInstance;

// Frame type used to control FSM

#define CP_NOFRAME 2000
#define CP_UPLOAD 2001
#define CP_DATA 2002
#define CP_FINAL 2003

// Subframe type used to control FSM

#define CPB_NONE 1000
#define CPB_FRAMESTART 1001
#define CPB_UIFRAME 1002
#define CPB_MSBFRAMELENGTH 1003
#define CPB_LSBFRAMELENGTH 1004
#define CPB_STREAMID 1005
#define CPB_COMMAND 1006
#define CPB_NOP 1007
#define CPB_MSBDATALENGTH 1008
#define CPB_SMSBDATALENGTH 1009
#define CPB_SLSBDATALENGTH 1010
#define CPB_LSBDATALENGTH 1011
#define CPB_MSBBLOCKNUMBER 1012
#define CPB_LSBBLOCKNUMBER 1013
#define CPB_MSBBLOCKNUMBERC 1014
#define CPB_LSBBLOCKNUMBERC 1015
#define CPB_DATA 1016
#define CPB_ESCDATA 1017
```

```
#define CPB_EOT 1018
#define CPB_CRC1 1019
#define CPB_CRC2 1020
#define CPB_FRAMEEND 1021
#define CPB_FL0 1022

// Constants defined by IBM's communications protocol

#define CP_ESCAPE 0x7D
#define CP_FRAME_START 0xC0
#define CP_UI_FRAME 0xA
#define CP_GET_SET 0x3
#define CP_STREAM 2
#define CP_FRAME_END 0xC1
#define CP_NEGOTIATE_ID 0
#define CP_BYTE_VERB 2
#define CP_RESPONSE 0
#define CP_SUCCESS 0x65
#define CP_ACK 6
#define CP_NACK 0x15
#define CP_EOT 7
#define CP_BEGIN_STREAM 3000
#define CP_NOOP 0
#define CP_NOP 2

#define uWORD                       unsigned int
#define uBYTE                       unsigned char
#define initialCrcValue (uWORD)0xFFFF
#define goodCrcValue    (uWORD)0xF0B8

/*----------------------------------------------------------------------
-*/
/*  CRC-16 lookup table
*/
```

```
/*----------------------------------------------------------------------
-*/
const uWORD crcLookupTable[256]=
{
 0x0000,0x1189,0x2312,0x329b,0x4624,0x57ad,0x6536,0x74bf,
 0x8c48,0x9dc1,0xaf5a,0xbed3,0xca6c,0xdbe5,0xe97e,0xf8f7,
 0x1081,0x0108,0x3393,0x221a,0x56a5,0x472c,0x75b7,0x643e,
 0x9cc9,0x8d40,0xbfdb,0xae52,0xdaed,0xcb64,0xf9ff,0xe876,
 0x2102,0x308b,0x0210,0x1399,0x6726,0x76af,0x4434,0x55bd,
 0xad4a,0xbcc3,0x8e58,0x9fd1,0xeb6e,0xfae7,0xc87c,0xd9f5,
 0x3183,0x200a,0x1291,0x0318,0x77a7,0x662e,0x54b5,0x453c,
 0xbdcb,0xac42,0x9ed9,0x8f50,0xfbef,0xea66,0xd8fd,0xc974,
 0x4204,0x538d,0x6116,0x709f,0x0420,0x15a9,0x2732,0x36bb,
 0xce4c,0xdfc5,0xed5e,0xfcd7,0x8868,0x99e1,0xab7a,0xbaf3,
 0x5285,0x430c,0x7197,0x601e,0x14a1,0x0528,0x37b3,0x263a,
 0xdecd,0xcf44,0xfddf,0xec56,0x98e9,0x8960,0xbbfb,0xaa72,
 0x6306,0x728f,0x4014,0x519d,0x2522,0x34ab,0x0630,0x17b9,
 0xef4e,0xfec7,0xcc5c,0xddd5,0xa96a,0xb8e3,0x8a78,0x9bf1,
 0x7387,0x620e,0x5095,0x411c,0x35a3,0x242a,0x16b1,0x0738,
 0xffcf,0xee46,0xdcdd,0xcd54,0xb9eb,0xa862,0x9af9,0x8b70,
 0x8408,0x9581,0xa71a,0xb693,0xc22c,0xd3a5,0xe13e,0xf0b7,
 0x0840,0x19c9,0x2b52,0x3adb,0x4e64,0x5fed,0x6d76,0x7cff,
 0x9489,0x8500,0xb79b,0xa612,0xd2ad,0xc324,0xf1bf,0xe036,
 0x18c1,0x0948,0x3bd3,0x2a5a,0x5ee5,0x4f6c,0x7df7,0x6c7e,
 0xa50a,0xb483,0x8618,0x9791,0xe32e,0xf2a7,0xc03c,0xd1b5,
 0x2942,0x38cb,0x0a50,0x1bd9,0x6f66,0x7eef,0x4c74,0x5dfd,
 0xb58b,0xa402,0x9699,0x8710,0xf3af,0xe226,0xd0bd,0xc134,
 0x39c3,0x284a,0x1ad1,0x0b58,0x7fe7,0x6e6e,0x5cf5,0x4d7c,
 0xc60c,0xd785,0xe51e,0xf497,0x8028,0x91a1,0xa33a,0xb2b3,
 0x4a44,0x5bcd,0x6956,0x78df,0x0c60,0x1de9,0x2f72,0x3efb,
 0xd68d,0xc704,0xf59f,0xe416,0x90a9,0x8120,0xb3bb,0xa232,
 0x5ac5,0x4b4c,0x79d7,0x685e,0x1ce1,0x0d68,0x3ff3,0x2e7a,
 0xe70e,0xf687,0xc41c,0xd595,0xa12a,0xb0a3,0x8238,0x93b1,
 0x6b46,0x7acf,0x4854,0x59dd,0x2d62,0x3ceb,0x0e70,0x1ff9,
 0xf78f,0xe606,0xd49d,0xc514,0xb1ab,0xa022,0x92b9,0x8330,
 0x7bc7,0x6a4e,0x58d5,0x495c,0x3de3,0x2c6a,0x1ef1,0x0f78
};

/*
        Cleanup - Initialize communications variables for the instance
*/
void Cleanup(PluginInstance* This)
{
        This->dwInBufferCount=0;
        This->dwInBufferIndex=0;
        This->dwOutBufferCount=0;
        This->dwFrame=CP_NOFRAME;
        This->dwSubFrame=CPB_NONE;
        This->dwFrameLength=0;
        This->dwDataLength=0;
        This->dwBlockNumber=0;
        This->dwPreviousBlockNumber=-1;
        This->dwBlockNumberC=0;
        This->dwBlockStart=0;
        This->dwNackCount=0;
        This->gReading=FALSE;
}

/*------------------------------------------------------------------------
-*/
/*  CrcCalculate            Calculate a new CRC given the current
*/
/*                          CRC and the new data.
*/
/*------------------------------------------------------------------------
-*/
uWORD CrcCalculate
  ( uWORD oldCrc,          /* in: CRC calculated "so far" */
    uBYTE Data)         /* in: data byte to calculate CRC on */
{
   uWORD newCrc = oldCrc;

   newCrc = (oldCrc >> 8) ^ crcLookupTable[(oldCrc ^ Data) & 0xff];

   return newCrc;
}

/*-------------------------------------------------------------------------
-*
 *  calculateCrc          Calculate a new CRC given the current CRC and * the new data.
*


*-------------------------------------------------------------------------
*/
uWORD calculateCrc
   ( uWORD oldCrc,       /* in: CRC calculated "so far" */
     uBYTE* pData, /* in: data bytes to calculate CRC on */
     uWORD len)    /* in: number of data bytes */
  {
   register uWORD newCrc = oldCrc;

   while (len--)
     newCrc = (newCrc >> 8) ^ crcLookupTable[(newCrc ^ *pData++) & 0xff];

   return newCrc;

  } /* calculateCrc */

uWORD checkCrc(uWORD length, uBYTE * buffer)
 {
 uWORD CRC = initialCrcValue;
   CRC = calculateCrc(CRC, buffer, length);
```

```
if (CRC == goodCrcValue)
  return 1;
return 0;
}

void DoAck(HANDLE hComm) {
        unsigned char ackBuffer[9];
        DWORD dwWritten;
  ackBuffer[0] = CP_FRAME_START;
  ackBuffer[1] = CP_UI_FRAME;
  ackBuffer[2] = 0; //Length
  ackBuffer[3] = 1;
  ackBuffer[4] = CP_STREAM;
  ackBuffer[5] = CP_ACK;
  ackBuffer[6] = 0x85; //CRC 1
  ackBuffer[7] = 0x8F; //CRC 2
  ackBuffer[8] = CP_FRAME_END;
        WriteFile(hComm,&ackBuffer[0],1,&dwWritten,NULL);
        Sleep(10);
        WriteFile(hComm,&ackBuffer[0],1,&dwWritten,NULL);
        Sleep(10);
        WriteFile(hComm,&ackBuffer[0],1,&dwWritten,NULL);
        Sleep(10);
        WriteFile(hComm,&ackBuffer[0],1,&dwWritten,NULL);
        Sleep(10);
        WriteFile(hComm,&ackBuffer[0],1,&dwWritten,NULL);
        Sleep(10);
        WriteFile(hComm,&ackBuffer[1],1,&dwWritten,NULL);
        Sleep(10);
        WriteFile(hComm,&ackBuffer[2],1,&dwWritten,NULL);
        Sleep(10);
        WriteFile(hComm,&ackBuffer[3],1,&dwWritten,NULL);
        Sleep(10);
        WriteFile(hComm,&ackBuffer[4],1,&dwWritten,NULL);
        Sleep(10);
        WriteFile(hComm,&ackBuffer[5],1,&dwWritten,NULL);
        Sleep(10);
        WriteFile(hComm,&ackBuffer[6],1,&dwWritten,NULL);
        Sleep(10);
        WriteFile(hComm,&ackBuffer[7],1,&dwWritten,NULL);
        Sleep(10);
        WriteFile(hComm,&ackBuffer[8],1,&dwWritten,NULL);
}

void DoNack(HANDLE hComm) {
        unsigned char nackBuffer[9];
        DWORD dwWritten;
  nackBuffer[0] = CP_FRAME_START;
  nackBuffer[1] = CP_UI_FRAME;
  nackBuffer[2] = 0; //Length
  nackBuffer[3] = 1;
  nackBuffer[4] = CP_STREAM;
  nackBuffer[5] = CP_NACK;
  nackBuffer[6] = 0x9F; //CRC 1
  nackBuffer[7] = 0xAD; //CRC 2
  nackBuffer[8] = CP_FRAME_END;
        WriteFile(hComm,&nackBuffer[0],1,&dwWritten,NULL);
        Sleep(10);
        WriteFile(hComm,&nackBuffer[0],1,&dwWritten,NULL);
        Sleep(10);
        WriteFile(hComm,&nackBuffer[0],1,&dwWritten,NULL);
        Sleep(10);
        WriteFile(hComm,&nackBuffer[0],1,&dwWritten,NULL);
        Sleep(10);
        WriteFile(hComm,&nackBuffer[0],1,&dwWritten,NULL);
        Sleep(10);
        WriteFile(hComm,&nackBuffer[1],1,&dwWritten,NULL);
        Sleep(10);
```

```
          WriteFile(hComm,&nackBuffer[2],1,&dwWritten,NULL);
          Sleep(10);
          WriteFile(hComm,&nackBuffer[3],1,&dwWritten,NULL);
          Sleep(10);
          WriteFile(hComm,&nackBuffer[4],1,&dwWritten,NULL);
          Sleep(10);
          WriteFile(hComm,&nackBuffer[5],1,&dwWritten,NULL);
          Sleep(10);
          WriteFile(hComm,&nackBuffer[6],1,&dwWritten,NULL);
          Sleep(10);
          WriteFile(hComm,&nackBuffer[7],1,&dwWritten,NULL);
          Sleep(10);
          WriteFile(hComm,&nackBuffer[8],1,&dwWritten,NULL);
}

BYTE TranslateDigitHex(unsigned char b) {
  switch (b) {
  case 0 :
   return '0';
  case 1 :
   return '1';
  case 2 :
   return '2';
  case 3 :
   return '3';
  case 4 :
   return '4';
  case 5 :
   return '5';
  case 6 :
   return '6';
  case 7 :
   return '7';
  case 8 :
       return '8';
      case 9 :
       return '9';
      case 10 :
       return 'a';
      case 11 :
       return 'b';
      case 12 :
       return 'c';
      case 13 :
       return 'd';
      case 14 :
       return 'e';
      case 15 :
       return 'f';
      default:
       return '0';
      }
    }

int instr(char *str1, char *str2) {
        int i1=0;
        int i2=0;
        int l=strlen(str2);
        do {
                if (str1[i1++]==str2[i2++]) {
                        if (i2==l) return 1;
                } else {
                        if (str1[i1]=='\0') return 0;
                        i2=0;
                }
        } while (1);
}
```

```
void AddMessage( HWND hwnd, char* message )
{
        int i;

        if( gMessageTextIndex >= gNumLines )            // If exceeded preset line
number display, reset to first line.
        {
                // Clear array and resetcounter

                for (i = 0; i < gNumLines; i++ )
                {
                        strcpy( gMessageTextArray[i], "" );
                }

                gMessageTextIndex = 0;
        }

        strcpy( gMessageTextArray[gMessageTextIndex++], message );

        if( hwnd )         // So messages can be collected while a valid window
handle hasn't been declared.
        {
                InvalidateRect( hwnd, NULL, TRUE );
                UpdateWindow( hwnd );
        }
}

NPError NPP_Initialize(void) {
#ifdef  _DEBUG
        {
                char str[100];
                sprintf(str,"NPP_Initialize\r\n");
                OutputDebugString(str);
        }
```

```
#endif

        gConnected = 0;
        outBufferSize = 100000;
        inBufferSize = outBufferSize*2+2;
        inBuffer = (unsigned char *) NPN_MemAlloc(inBufferSize);
        outBuffer = (unsigned char *) NPN_MemAlloc(outBufferSize);
        if ((inBuffer==NULL) || (outBuffer==NULL)) {
                if (inBuffer) {
                        NPN_MemFree(inBuffer);
                        inBuffer = NULL;
                }
                if (outBuffer) {
                        NPN_MemFree(outBuffer);
                        outBuffer = NULL;
                }
                return NPERR_OUT_OF_MEMORY_ERROR;
        }
  return NPERR_NO_ERROR;
}

jref NPP_GetJavaClass(void) {
        return NULL;
}

// Deallocate I/O buffers and close the COM port void NPP_Shutdown(void) {

        // Close the comm connection;

#ifdef  _DEBUG
        {
                char str[100];
                sprintf(str,"NPP_Shutdown gConnected=%d
hComm=%8.8lx\r\n",gConnected,hComm);
```

```
                OutputDebugString(str);
        }
#endif

        if (gConnected)
                CloseHandle(hComm);
        gConnected=0;

        // Free memory.

        if (inBuffer!=NULL) NPN_MemFree(inBuffer);
        inBuffer=NULL;
        if (outBuffer!=NULL) NPN_MemFree(outBuffer);
        outBuffer=NULL;
}

NPError NPP_New(NPMIMEType pluginType,
        NPP instance,
        uint16 mode,
        int16 argc,
        char* argn[],
        char* argv[],
        NPSavedData* saved) {

        DCB dcb;
        COMMTIMEOUTS ctm;
        BOOL gSuccess;
        int i;
        NPError result = NPERR_NO_ERROR;
        PluginInstance* This;

#ifdef  _DEBUG
        {
                char str[100];
                sprintf(str,"NPP_New instance=%8.8lx
gConn=%d\r\n",instance,gConnected);
                OutputDebugString(str);
        }
#endif

        if (instance == NULL) {
                return NPERR_INVALID_INSTANCE_ERROR;
        }
        instance->pdata = NPN_MemAlloc(sizeof(PluginInstance));
        This = (PluginInstance*) instance->pdata;
        if (This == NULL) {
          return NPERR_OUT_OF_MEMORY_ERROR;
        }
        /* mode is NP_EMBED, NP_FULL, or NP_BACKGROUND (see npapi.h) */
        This->fWindow = NULL;
        This->fMode = mode;
        This->fhWnd = NULL;
        This->fDefaultWindowProc = NULL;

        // Initialize communications variables
        Cleanup(This);

        // Save plug-in instance

        This->gInstance = instance;

        // Get plugin parameters (hostname,hostport,uid,proxyname,proxyport,
        // comm port, baud rate, sourceurl) that
        // was passed into the plugin via html.

        This->gHostName[0] = '\0';
        This->gHostPort[0] = '\0';
        This->gUID[0] = '\0';
```

```
This->gProxyName[0] = '\0';
This->gProxyPort[0] = '\0';
This->gComPort[0] = '\0';
This->gComSpeed[0] = '\0';
This->gSourceURL[0] = '\0';
This->gVerbose = FALSE;
This->gVersion[0] = '\0';

for (i=0; i<argc; i++) {
        if (strcmp(strupr(argn[i]),"HOSTNAME")==0) {
                strcpy( This->gHostName,        argv[i]);

        } else if (strcmp(strupr(argn[i]),"HOSTPORT")==0) {
                strcpy( This->gHostPort,        argv[i]);
        } else if (strcmp(strupr(argn[i]),"UID")==0) {
                strcpy( This->gUID,     argv[i]);
        } else if (strcmp(strupr(argn[i]),"PROXYNAME")==0) {
                strcpy( This->gProxyName,       argv[i]);
        } else if (strcmp(strupr(argn[i]),"PROXYPORT")==0) {
                strcpy( This->gProxyPort,       argv[i]);
        } else if (strcmp(strupr(argn[i]),"COMPORT")==0) {
                strcpy( This->gComPort,         argv[i]);
        } else if (strcmp(strupr(argn[i]),"COMSPEED")==0) {
                strcpy( This->gComSpeed,        argv[i]);
        } else if (strcmp(strupr(argn[i]),"SOURCEURL")==0) {
                strcpy( This->gSourceURL,       argv[i]);
        } else if (strcmp(strupr(argn[i]),"NUMLINES")==0) {
                gNumLines=atoi( argv[i] );
        } else if (strcmp(strupr(argn[i]),"VERBOSE")==0) {
                This->gVerbose=TRUE;
        } else if (strcmp(strupr(argn[i]),"VERSION")==0) {
                strcpy( This->gVersion, argv[i]);
        }
}
```

```
        // Close the comm connection so that the port parameters can be
reset

        if (gConnected)         {
                CloseHandle(hComm);
        }

#ifdef  _DEBUG
        {
                char str[100];
                sprintf(str,"Closed comm port instance=%8.8lx
gConn=%d\r\n",instance,gConnected);
                OutputDebugString(str);
        }
#endif
        // Connect to the Comm port and allocate the buffers.

//      hComm=CreateFile("D:\\TEMP\\Copy (2) of COMMLOG.BIN",GENERIC_READ |
GENERIC_WRITE ,FILE_SHARE_WRITE,NULL,OPEN_EXISTING,0,NULL);
        hComm=CreateFile(This->gComPort,GENERIC_READ | GENERIC_WRITE
,FILE_SHARE_WRITE,NULL,OPEN_EXISTING,0,NULL);
        if (hComm==INVALID_HANDLE_VALUE) {
                char message[256];
                strcpy( message, "Error connecting to ");
                strcat( message, This->gComPort );
                strcat( message, " - please confirm that it is available" );
                AddMessage( This->fhWnd, message ); // *****
                return 0;
        }
        ++gConnected;

#ifdef  _DEBUG
        {
                char str[100];
```

```
                    sprintf(str,"Opening hComm=%8.8lx
gConn=%d\r\n",hComm,gConnected);
                    OutputDebugString(str);
            }
#endif

          gSuccess=GetCommState(hComm,&dcb);
          if (!gSuccess) {
                    AddMessage( This->fhWnd, "Error on GetCommState()..." );
// *****
                    return 0;
          }
          dcb.DCBlength=sizeof(dcb);
          dcb.BaudRate=atol( This->gComSpeed );
          dcb.ByteSize=8;
          dcb.Parity=NOPARITY;
          dcb.StopBits=ONESTOPBIT;
          dcb.fBinary=1;
          gSuccess=SetCommState(hComm,&dcb);
          if (!gSuccess) {
                    AddMessage( This->fhWnd, "Error on GetCommState()..." );
// *****
                    return 0;
          }
          ctm.ReadIntervalTimeout=MAXDWORD;
          ctm.ReadTotalTimeoutConstant=0;
          ctm.ReadTotalTimeoutMultiplier=0;
          ctm.WriteTotalTimeoutConstant=0;
          ctm.WriteTotalTimeoutMultiplier=0;
          gSuccess=SetCommTimeouts(hComm,&ctm);
          if (!gSuccess) {
                    AddMessage( This->fhWnd, "Error on SetCommTimeouts()..." );
// *****
                    return 0;
```

```
        }
        {
        char message[256];
        sprintf(message,"Connected to %s - please initiate upload from
pad...",This->gComPort);
        AddMessage( This->fhWnd, message ); // *****
        }
        Cleanup(This);

        // Check the version

        if (strcmp(This->gVersion,"1.2.6")!=0) {
                AddMessage( This->fhWnd, "Warning - incorrect version of
plug-in is installed.  Please upgrade plug-in..." );        // *****
        }

#ifdef  _DEBUG
        {
                char str[100];
                sprintf(str,"End of NPP_New instance=%8.8lx
gConn=%d\r\n",instance,gConnected);
                OutputDebugString(str);
        }
#endif

        return result;
}

NPError NPP_Destroy(NPP instance, NPSavedData** save) {
        PluginInstance* This;

#ifdef  _DEBUG
        {
                char str[100];
                sprintf(str,"NPP_Destroy instance=%8.8lx
gCon=%d\r\n",instance,gConnected);
                OutputDebugString(str);
        }
#endif

        if (instance == NULL)
                return NPERR_INVALID_INSTANCE_ERROR;

        This = (PluginInstance*) instance->pdata;
        if (This != NULL) {
                // Kill the timer.

#ifdef  _DEBUG
                {
                        char str[100];
                        sprintf(str,"Destroy timer %8.8lx\r\n",This->fhWnd);
                        OutputDebugString(str);
                }
#endif

                KillTimer(This->fhWnd, 1);

                if( This->fWindow != NULL ) {
                        SetWindowLong( This->fhWnd, GWL_WNDPROC,
(LONG)This->fDefaultWindowProc);
                        This->fDefaultWindowProc = NULL;
                        This->fhWnd = NULL;
                }

                NPN_MemFree(instance->pdata);
                instance->pdata = NULL;
        }
        // Close the comm connection on the last instance only

        if (gConnected == 1)
        {       CloseHandle(hComm);

        }
        --gConnected;

        return NPERR_NO_ERROR;
}

NPError NPP_SetWindow(NPP instance, NPWindow* window) {
        NPError result = NPERR_NO_ERROR;
        PluginInstance* This;
        HWND hButton;
        HANDLE hImage;
        HANDLE hInstance;
        RECT rect;

        if (instance == NULL)
                return NPERR_INVALID_INSTANCE_ERROR;

        This = (PluginInstance*) instance->pdata;

#ifdef  _DEBUG
        {
                char str[100];
                sprintf(str,"NPP_SetWindow
instance=%8.8lx(%8.8lx)\r\n",instance,This->gInstance);
                OutputDebugString(str);
        }
#endif

        if( This->fWindow != NULL ) /* If we already have a window, clean
```

```
                                                    * it up
before trying to subclass
                                                    * the new
window. */
        {
                if( (window == NULL) || ( window->window == NULL ) ) {
                        /* There is now no window to use. get rid of the old
                         * one and exit. */
                        SetWindowLong( This->fhWnd, GWL_WNDPROC,
(LONG)This->fDefaultWindowProc);
                        This->fDefaultWindowProc = NULL;
                        This->fhWnd = NULL;
                        This->fWindow=window;
                        return NPERR_NO_ERROR;
                }

                else if ( This->fhWnd == (HWND) window->window ) {
                        /* The new window is the same as the old one. Redraw
and get out. */
                        InvalidateRect( This->fhWnd, NULL, TRUE );
                        UpdateWindow( This->fhWnd );
                        This->fWindow=window;
                        return NPERR_NO_ERROR;
                }
                else {
                        /* Clean up the old window, so that we can subclass
the new
                         * one later. */
                        SetWindowLong( This->fhWnd, GWL_WNDPROC,
(LONG)This->fDefaultWindowProc);
                        This->fDefaultWindowProc = NULL;
                        This->fhWnd = NULL;
                }
        }
        else if( (window == NULL) || ( window->window == NULL ) ) {
                /* We can just get out of here if there is no current
                 * window and there is no new window to use. */
                        This->fWindow=window;
                return NPERR_NO_ERROR;
        }

        /* At this point, we will subclass
         * window->window so that we can begin drawing and
         * receiving window messages. */
#ifdef  _DEBUG
        {
                char str[200];
                sprintf(str,"Subclassing window %8.8lx  fhWnd =
%8.8lx\r\n",window->window,This->fhWnd);
                OutputDebugString(str);
        }
#endif

        This->fDefaultWindowProc = (WNDPROC)SetWindowLong(
(HWND)window->window, GWL_WNDPROC, (LONG)PluginWindowProc);
        This->fhWnd = (HWND) window->window;
        SetProp( This->fhWnd, gInstanceLookupString, (HANDLE)This);

        try
        {
                IApplicationPtr pApp(__uuidof(Application));

                This->bTransNote = TRUE;
        }
        catch(...)
        {
                This->bTransNote = FALSE;
        }
```

```
// Create button
        GetClientRect(This->fhWnd,&rect);
        gNumLines = rect.bottom/20;
        if (This->bTransNote)
        {       hInstance = (HANDLE)
GetWindowLong(This->fhWnd,GWL_HINSTANCE);
                hButton = CreateWindow("button","IBM Upload",WS_CHILD |
WS_BORDER | WS_VISIBLE | BS_PUSHBUTTON | BS_CENTER | BS_BITMAP |
BS_VCENTER,
                                        rect.right-120,0,120,32,
                                        This->fhWnd,(HMENU) 1,(HINSTANCE)
hInstance,NULL);
                hImage =
LoadImage(GetModuleHandle("NPTimbrl.dll"),MAKEINTRESOURCE(IDB_WORKONCE),IMAG
E_BITMAP,0,0,LR_SHARED);
                if (hImage)
                        SendMessage(hButton,BM_SETIMAGE,IMAGE_BITMAP,(LONG)
hImage);
        }
#ifdef  _DEBUG
        else
        {       hInstance = (HANDLE)
GetWindowLong(This->fhWnd,GWL_HINSTANCE);
                hButton = CreateWindow("button","Upload File",WS_CHILD |
WS_BORDER | WS_VISIBLE | BS_PUSHBUTTON | BS_CENTER | BS_VCENTER,
                                        rect.right-90,0,90,30,
                                        This->fhWnd,(HMENU) 1,(HINSTANCE)
hInstance,NULL);
        }
#endif

//      Create timer for window
#ifdef  _DEBUG
        {
```

```
                    char str[100];
                    sprintf(str,"Create timer %8.8lx\r\n",This->fhWnd);
                    OutputDebugString(str);
          }
#endif

          SetTimer( This->fhWnd, 1, 0, NULL );

          InvalidateRect( This->fhWnd, NULL, TRUE );
          UpdateWindow( This->fhWnd );

          This->fWindow = window;
          return result;
}

NPError NPP_NewStream(NPP instance,
              NPMIMEType type,
              NPStream *stream,
              NPBool seekable,
              uint16 *stype) {
          PluginInstance* This;

          if (instance == NULL)
                    return NPERR_INVALID_INSTANCE_ERROR;

          This = (PluginInstance*) instance->pdata;

          return NPERR_NO_ERROR;
}

int32 STREAMBUFSIZE = 0X0FFFFFFF; /* If we are reading from a file in
NPAsFile
                                                              * mode so
we can take any size stream in our
```

```
                                                            * write
call (since we ignore it) */

int32 NPP_WriteReady(NPP instance, NPStream *stream) {
        PluginInstance* This;
        if (instance != NULL)
                This = (PluginInstance*) instance->pdata;
        return STREAMBUFSIZE;
}

int32 NPP_Write(NPP instance, NPStream *stream, int32 offset, int32 len,
void *buffer) {
        if (instance != NULL) {
                PluginInstance* This = (PluginInstance*) instance->pdata;
        }
        return len;             /* The number of bytes accepted */
}

NPError NPP_DestroyStream(NPP instance, NPStream *stream, NPError reason) {
        PluginInstance* This;

        if (instance == NULL)
                return NPERR_INVALID_INSTANCE_ERROR;
        This = (PluginInstance*) instance->pdata;

        return NPERR_NO_ERROR;
}

void NPP_StreamAsFile(NPP instance, NPStream *stream, const char* fname) {
        PluginInstance* This;
        if (instance != NULL)
                This = (PluginInstance*) instance->pdata;
}
```

```
void NPP_Print(NPP instance, NPPrint* printInfo) {
        if(printInfo == NULL)
                return;

        if (instance != NULL) {
                PluginInstance* This = (PluginInstance*) instance->pdata;

                if (printInfo->mode == NP_FULL) {

                        void* platformPrint =
                                printInfo->print.fullPrint.platformPrint;
                        NPBool printOne =
                                printInfo->print.fullPrint.printOne;

                        /* Do the default*/
                        printInfo->print.fullPrint.pluginPrinted = FALSE;
                }
                else {  /* If not fullscreen, we must be embedded */
                        NPWindow* printWindow =
                                &(printInfo->print.embedPrint.window);
                        void* platformPrint =
                                printInfo->print.embedPrint.platformPrint;
                }
        }
}

void NPP_URLNotify( NPP instance, const char* url, NPReason reason, void*
notifyData ) {
        switch( reason ) {
        case NPRES_DONE:              // Completed normally.
                break;
        case NPRES_USER_BREAK:  // User canceled stream directly or
indirectly.
                break;
        case NPRES_NETWORK_ERR: // Stream failed due to problems with
network, disk I/O, lack of memory, or other problems.
                break;
        }
}

int16 NPP_HandleEvent(NPP instance, void* event)
{
        return 0;
}

int PostURL(HWND hWnd, char *hostname, unsigned short hostport, int uid,
unsigned char* buffer, int bufferlen, char *proxyname, unsigned short
proxyport) {
        SOCKET skt;
        INT iResult;
        SOCKADDR_IN server;
        WSADATA wsaData;
        HOSTENT *host;
        BOOL useproxy=0;
        int cc=0;
        char httpBuffer[256];
        PluginInstance* This = (PluginInstance*) GetProp(hWnd,
gInstanceLookupString);

        iResult = WSAStartup(0x202,&wsaData);
        if (iResult==SOCKET_ERROR) {
                sprintf(httpBuffer,"Error on %d
WSAStartup()...",WSAGetLastError());
                AddMessage(hWnd,httpBuffer);
                return -1;
        }
        skt=socket(AF_INET,SOCK_STREAM,0);
        if (skt<0) {
```

```
			sprintf(httpBuffer,"Error %d on
socket()...",WSAGetLastError());
			AddMessage(hWnd,httpBuffer);
			return -1;
		}
		if (strcmp(proxyname,"")!=0) useproxy=1;
		if (useproxy==1) {
			sprintf(httpBuffer,"Looking up proxy %s...",proxyname);
			if (This->gVerbose) AddMessage(hWnd,httpBuffer);
			host=gethostbyname(proxyname);
			server.sin_port=htons(proxyport);
		} else {
			sprintf(httpBuffer,"Looking up host %s...",hostname);
			if (This->gVerbose) AddMessage(hWnd,httpBuffer);
			host=gethostbyname(hostname);
			server.sin_port=htons(hostport);
		}
		if (host==NULL) {
			sprintf(httpBuffer,"Error %d on
gethostbyname()...",WSAGetLastError());
			AddMessage(hWnd,httpBuffer);
			return -1;
		}
		memcpy(&(server.sin_addr),*host->h_addr_list,host->h_length);
		server.sin_family=host->h_addrtype;
		if (This->gVerbose) AddMessage(hWnd,"Connecting...");
		//iResult=connect(skt,(SOCKADDR*)&server,sizeof(server));
		//if (iResult==SOCKET_ERROR) {
		//	sprintf(httpBuffer,"Error %d on
connect()...",WSAGetLastError());
		//	AddMessage(hWnd,httpBuffer);
		//	return -1;
		//}
		// Try connecting multiple times - this exists to help us manage
```

```
        // peak OCRServer traffic while we scale.  If no processes
        // are available to service this connection, try again a number of
        // times.
        for (cc=0;cc<10;cc++) {
                iResult=connect(skt,(SOCKADDR*)&server,sizeof(server));
                if (iResult!=SOCKET_ERROR) break;
                Sleep(100);
        }
        if (cc==10) {
                sprintf(httpBuffer,"Server Busy - Please Try Again...");
                AddMessage(hWnd,httpBuffer);
                return -1;
        }
        if (This->gVerbose) AddMessage(hWnd,"Executing HTTP POST
method...");
        if (useproxy==1) {
                if (hostport!=80) {
                        sprintf(httpBuffer,"POST http://%s:%d/%d/
HTTP/1.0\nContent-Type: application/x-www-form-urlencoded\nContent-Length:
%d\n\n",hostname,hostport,uid,bufferlen);
                } else {
                        sprintf(httpBuffer,"POST http://%s/%d/
HTTP/1.0\nContent-Type: application/x-www-form-urlencoded\nContent-Length:
%d\n\n",hostname,uid,bufferlen);
                }
        } else {
                sprintf(httpBuffer,"POST /%d \nContent-Type:
application/x-www-form-urlencoded\nContent-Length: %d\n\n",uid,bufferlen);
        }
        iResult=send(skt,(const char*)httpBuffer,strlen(httpBuffer),0);
        iResult=send(skt,(const char*)buffer,bufferlen,0);
        if (This->gVerbose) AddMessage(hWnd,"Waiting on HTTP response...");
        iResult=recv(skt,httpBuffer,sizeof(httpBuffer),0);
        closesocket(skt);
```

```
		WSACleanup();

		if (This->gVerbose) AddMessage(hWnd,"Socket closed...");
		if (iResult==SOCKET_ERROR) {
			AddMessage(hWnd,"Error on recv()...");
			return -1;
		} else if (iResult==0) {
			AddMessage(hWnd,"Error on recv()...");
			return -1;
		} else {
			httpBuffer[iResult]='\0';
			if (This->gVerbose) AddMessage(hWnd,"Received HTTP
	response...");
		}
		if (instr(httpBuffer,"<body>OK</body>")==1) {
			return 0;
		} else {
			if (This->gVerbose) AddMessage(hWnd,httpBuffer);
			return -1;
		}
	}

	char* AddTick(char *str) {
		static int tickCount;
		int i;
		if (str==NULL) {
			tickCount=0;
			return NULL;
		} else {
			tickCount++;
			sprintf(str,"Uploading");
			for (i=0;i<tickCount;i++)
				str[9+i]='.';
			str[9+i]='\0';
```

```
                return str;
        }
}

uBYTE GetNextByte(unsigned char *inBuffer,DWORD *index)
{
        uBYTE result;

        result = inBuffer[*index];
        ++*index;
        if (result == CP_ESCAPE) {
                result = inBuffer[*index] ^ 0x20;
                ++*index;
        }
        return result;
}

LRESULT CALLBACK PluginWindowProc( HWND hWnd, UINT Msg, WPARAM wParam,
LPARAM lParam)
{
        PluginInstance* This = (PluginInstance*) GetProp(hWnd,
gInstanceLookupString);

        PAINTSTRUCT paintStruct;
        HDC                     hdc;

        //static unsigned char inBuffer[150000];
        //static unsigned char outBuffer[75000];
        DWORD dwRead;
        BOOL fDone=FALSE;
        char message[256];
        static uWORD CRC=0;
        unsigned char b2=0;
        DWORD i2=0;
        DWORD i1=0;
        NPError err;
        int i;
        BOOL gSuccess;
        HWND hButton;
        RECT rect;
        long w,h;
        HANDLE hFile;
        _bstr_t szFileName;

        switch( Msg ) {
                case WM_SIZE:
                        hButton = GetDlgItem(hWnd,1);
                        if (IsWindow(hButton))
                        {       GetWindowRect(hButton,&rect);
                                w = rect.right-rect.left;
                                h = rect.bottom-rect.top;
                                GetClientRect(hWnd,&rect);
#ifdef  _DEBUG
                                {
                                        char str[100];
                                        sprintf(str,"Rect (%d,%d)-(%d,%d)
Button  %d x %d\r\n",

rect.left,rect.top,rect.right,rect.bottom,w,h);
                                        OutputDebugString(str);
                                }
#endif
                                MoveWindow(hButton,rect.right-w,0,w,h,TRUE);
                        }
                        GetClientRect(hWnd,&rect);
                        gNumLines = rect.bottom/20;
                        break;
                case WM_COMMAND:
                    if (This->bTransNote)
                    {       try
                            {
                                    IApplicationPtr
pApp(__uuidof(Application));
                                    IArchivePtr    pArchive =
pApp->Archive;
//                                  long lPadNo = pArchive->Count;
//                                  IPadInfoPtr    pPadInfo =
pArchive->Item[(long) (lPadNo-1)];
                                    IPadInfoPtr    pPadInfo =
pArchive->GetActivePad();
                                    szFileName = pPadInfo->FileName;
                            }
                            catch(...)
                            {
                                    AddMessage( This->fhWnd,"Error
accessing COM object");
                                    break;
                            }
                    }
#ifdef   _DEBUG
                    else
                    {       OPENFILENAME ofn;

                            memset(&ofn,0,sizeof(ofn));
                            ofn.lStructSize = sizeof(ofn);
                            ofn.hwndOwner = hWnd;
                            ofn.lpstrFilter = "Ink Files
(*.ixu,*.pad)\0*.ixu;*.pad\0";
                            ofn.nFilterIndex = 1;
                            message[0] = 0;
                            ofn.lpstrFile = message;
                            ofn.nMaxFile = 256;
```

```
                                        ofn.Flags = OFN_ENABLESIZING | OFN_EXPLORER
| OFN_FILEMUSTEXIST;
                                        if (GetOpenFileName(&ofn))
                                                szFileName = ofn.lpstrFile;
                                        else
                                                break;
                                }
#endif
                                hFile =
CreateFile(szFileName,GENERIC_READ,FILE_SHARE_READ,NULL,OPEN_EXISTING,0,NULL
);
                                if (INVALID_HANDLE_VALUE != hFile)
                                {       This->dwOutBufferCount =
GetFileSize(hFile,NULL);
                                        if (This->dwOutBufferCount > outBufferSize)
{
                                                unsigned char *tmp;

                                                if (This->gVerbose) {
                                                        AddMessage(
This->fhWnd,"Increasing buffer size");
                                                }

                                                tmp = (unsigned char *)
NPN_MemAlloc(This->dwDataLength);
                                                if (tmp == NULL) {
                                                        AddMessage(This->fhWnd,
"Unable to reallocate output buffer.");
                                                        Cleanup(This);
                                                }
                                                else {

memcpy(tmp,outBuffer,outBufferSize);
                                                        outBufferSize =
This->dwDataLength;
                                                            NPN_MemFree(outBuffer);
                                                            outBuffer = tmp;
                                                    }

                                                    tmp = (unsigned char *)
NPN_MemAlloc(This->dwDataLength*2+2);
                                                    if (tmp == NULL) {
                                                            AddMessage(This->fhWnd,
"Unable to reallocate input buffer.");
                                                            Cleanup(This);
                                                    }
                                                    else {

memcpy(tmp,inBuffer,inBufferSize);
                                                            inBufferSize =
This->dwDataLength*2+2;
                                                            NPN_MemFree(inBuffer);
                                                            inBuffer = tmp;
                                                    }
                                            }

ReadFile(hFile,outBuffer,This->dwOutBufferCount,(unsigned long *) &w,NULL);
                                            CloseHandle(hFile);
                                            sprintf(message,"Read %d bytes from
%s",This->dwOutBufferCount,This->gComPort);
                                            if (This->gVerbose) AddMessage( This->fhWnd,
message );        // *****
                                            strcpy((char *) inBuffer,"d=");
                                            i2=2;
                                            for (; i1<This->dwOutBufferCount;
i1++,i2+=2) {
                                                    b2=(unsigned
char)(outBuffer[i1]>>4);
```

```
                                        inBuffer[i2]=TranslateDigitHex(b2);

inBuffer[i2+1]=TranslateDigitHex((unsigned char)(outBuffer[i1]-(b2<<4)));
                                }

                                err = PostURL(hWnd,This->gHostName,(unsigned
short)atoi(This->gHostPort),atoi(This->gUID),inBuffer,This->dwOutBufferCount
*2+2,This->gProxyName,(unsigned short)atoi(This->gProxyPort));
                                if (err==0) {
                                        AddMessage( This->fhWnd,"Upload
Successful - please wait...");

NPN_GetURL(This->gInstance,This->gSourceURL,"_current");
                                } else {
                                        AddMessage( This->fhWnd,"Upload
Failed");
                                        Cleanup(This);
                                        fDone=FALSE;
                                        CRC=0;
                                }
                        }
                        break;
                case WM_TIMER:
                        do {

gSuccess=ReadFile(hComm,&inBuffer[This->dwInBufferCount],256,&dwRead,NULL);
                                if (!gSuccess) {
                                        i = GetLastError();
                                }
                                if (dwRead>0) {
                                        This->dwInBufferCount+=dwRead;
                                }
                                if
(This->dwInBufferIndex<This->dwInBufferCount) {
```

```
                                        // If escape char is at end of
buffer, wait for more data
                                        if ((inBuffer[This->dwInBufferIndex]
== CP_ESCAPE) && (This->dwInBufferIndex == This->dwInBufferCount-1))
                                              continue;
                                        switch (This->dwFrame) {
                                        case CP_NOFRAME :
                                              switch (This->dwSubFrame) {
                                              case CPB_NONE :
                                                    if
(inBuffer[This->dwInBufferIndex]==CP_FRAME_START) {

This->dwInBufferIndex++;

This->dwSubFrame=CPB_UIFRAME;
                                                    } else {

This->dwInBufferIndex++;
                                                    }
                                                    break;
                                              case CPB_UIFRAME :
                                                    if
(inBuffer[This->dwInBufferIndex]==CP_UI_FRAME) {
                                                          CRC =
initialCrcValue;
                                                          CRC =
CrcCalculate(CRC, GetNextByte(inBuffer,&This->dwInBufferIndex));

This->dwSubFrame=CPB_MSBFRAMELENGTH;
                                                    } else {
                                                          AddMessage(
This->fhWnd,"ERROR ONE..");

Cleanup(This);
```

```
                                                            return 0;
                                                    }
                                                    break;
                                            case CPB_MSBFRAMELENGTH :
                                                    CRC =
CrcCalculate(CRC, b2 = GetNextByte(inBuffer,&This->dwInBufferIndex));
                                                    This->dwFrameLength
= b2;

This->dwSubFrame=CPB_LSBFRAMELENGTH;
                                                    break;
                                            case CPB_LSBFRAMELENGTH :
                                                    CRC =
CrcCalculate(CRC, b2 = GetNextByte(inBuffer,&This->dwInBufferIndex));
                                                    This->dwFrameLength
= (This->dwFrameLength << 8) + b2;

This->dwSubFrame=CPB_STREAMID;
                                                    break;
                                            case CPB_STREAMID :
                                                    if
(inBuffer[This->dwInBufferIndex]==CP_STREAM) {
                                                            CRC =
CrcCalculate(CRC, GetNextByte(inBuffer,&This->dwInBufferIndex));

This->dwSubFrame=CPB_COMMAND;
                                                    } else {
                                                            AddMessage(
This->fhWnd,"Pad error - please erase pad and retry..");

Cleanup(This);
                                                            return 0;
                                                    }
                                                    break;
```

```
                    case CPB_COMMAND :
                        if
(inBuffer[This->dwInBufferIndex]==CP_NOP) {

This->dwFrame=CP_UPLOAD;

This->dwSubFrame=CPB_NOP;
                        } else if
(inBuffer[This->dwInBufferIndex]==CP_EOT) {

This->dwFrame=CP_FINAL;

This->dwSubFrame=CPB_EOT;
                        } else {

This->dwFrame=CP_DATA;

This->dwSubFrame=CPB_MSBBLOCKNUMBER;
                        }
                        break;
                    }
                    break;
                case CP_UPLOAD :
                    switch (This->dwSubFrame) {
                    case CPB_NOP :
                        CRC =
CrcCalculate(CRC, GetNextByte(inBuffer,&This->dwInBufferIndex));

This->dwSubFrame=CPB_MSBDATALENGTH;
                        break;
                    case CPB_MSBDATALENGTH :
                        CRC =
CrcCalculate(CRC, b2 = GetNextByte(inBuffer,&This->dwInBufferIndex));
                        This->dwDataLength =
b2;

This->dwSubFrame=CPB_SMSBDATALENGTH;
                                        break;
                                case CPB_SMSBDATALENGTH :
                                        CRC =
CrcCalculate(CRC, b2 = GetNextByte(inBuffer,&This->dwInBufferIndex));
                                        This->dwDataLength =
(This->dwDataLength << 8) + b2;

This->dwSubFrame=CPB_SLSBDATALENGTH;
                                        break;
                                case CPB_SLSBDATALENGTH :
                                        CRC =
CrcCalculate(CRC, b2 = GetNextByte(inBuffer,&This->dwInBufferIndex));
                                        This->dwDataLength =
(This->dwDataLength << 8) + b2;

This->dwSubFrame=CPB_LSBDATALENGTH;
                                        break;
                                case CPB_LSBDATALENGTH :
                                        CRC =
CrcCalculate(CRC, b2 = GetNextByte(inBuffer,&This->dwInBufferIndex));
                                        This->dwDataLength =
(This->dwDataLength << 8) + b2;

This->dwSubFrame=CPB_CRC1;
                                        break;
                                case CPB_CRC1 :
                                        CRC =
CrcCalculate(CRC, GetNextByte(inBuffer,&This->dwInBufferIndex));

This->dwSubFrame=CPB_CRC2;
                                        break;
                                        case CPB_CRC2 :
                                             CRC =
CrcCalculate(CRC, GetNextByte(inBuffer,&This->dwInBufferIndex));

This->dwSubFrame=CPB_FRAMEEND;
                                             break;
                                        case CPB_FRAMEEND :
                                             gSuccess = TRUE;
                                             if
(inBuffer[This->dwInBufferIndex]!=CP_FRAME_END) {
                                                  if
(This->gVerbose) {

AddMessage( This->fhWnd,"Frame End Offset...");
                                                  }
                                                  gSuccess =
FALSE;
                                             } else if
(CRC!=goodCrcValue) {
                                                  AddMessage(
This->fhWnd,"ERROR FOUR...");
                                                  gSuccess =
FALSE;
                                             }
                                             if (!gSuccess) {

This->dwSubFrame=CPB_NONE;

This->dwFrame=CP_NOFRAME;

DoNack(hComm);
                                                  if
(This->gVerbose) AddMessage( This->fhWnd,"NACK...");
```

```
    This->dwNackCount++;
                                        if
    (This->dwNackCount==3) fDone=TRUE;
                                        break;
                                    }
    This->dwInBufferIndex++;
    This->dwSubFrame=CPB_NONE;
    This->dwFrame=CP_NOFRAME;
                                    DoAck(hComm);
                                    if (This->gVerbose)
    {
                                        AddMessage(
    This->fhWnd,"ACK...");
                                    } else {
                                        AddMessage(
    This->fhWnd,AddTick(message));
                                    }
                                    This->dwNackCount=0;
                                    This->gReading=TRUE;
                                    if
    (This->dwDataLength > outBufferSize) {
                                        unsigned
    char *tmp;
                                        if
    (This->gVerbose) {
    AddMessage( This->fhWnd,"Increasing buffer size");
                                        }
```

```
tmp =
(unsigned char *) NPN_MemAlloc(This->dwDataLength);
if (tmp ==
NULL) {
AddMessage(This->fhWnd, "Unable to reallocate output buffer.");
Cleanup(This);
}
else {
memcpy(tmp,outBuffer,outBufferSize);
outBufferSize = This->dwDataLength;
NPN_MemFree(outBuffer);
outBuffer = tmp;
}
tmp =
(unsigned char *) NPN_MemAlloc(This->dwDataLength*2+2);
if (tmp ==
NULL) {
AddMessage(This->fhWnd, "Unable to reallocate input buffer.");
Cleanup(This);
}
else {
memcpy(tmp,inBuffer,inBufferSize);
inBufferSize = This->dwDataLength*2+2;
NPN_MemFree(inBuffer);

inBuffer = tmp;
                                                            }
                                                        }
                                                        break;
                                                }

                                                break;
                                        case CP_DATA :
                                                switch (This->dwSubFrame) {
                                                case CPB_MSBBLOCKNUMBER :
                                                        CRC =
CrcCalculate(CRC, b2 = GetNextByte(inBuffer,&This->dwInBufferIndex));
                                                        This->dwBlockNumber
= b2;

This->dwFrameLength--;

This->dwDataLength--;

This->dwSubFrame=CPB_LSBBLOCKNUMBER;
                                                        break;
                                                case CPB_LSBBLOCKNUMBER :
                                                        CRC =
CrcCalculate(CRC, b2 = GetNextByte(inBuffer,&This->dwInBufferIndex));
                                                        This->dwBlockNumber
= (This->dwBlockNumber << 8) + b2;

This->dwFrameLength--;

This->dwDataLength--;
```

```
This->dwSubFrame=CPB_MSBBLOCKNUMBERC;
                                                            break;
                                                    case CPB_MSBBLOCKNUMBERC :
                                                            CRC =
CrcCalculate(CRC, b2 = GetNextByte(inBuffer,&This->dwInBufferIndex));
                                                            This->dwBlockNumberC
= b2;

This->dwFrameLength--;

This->dwDataLength--;

This->dwSubFrame=CPB_LSBBLOCKNUMBERC;
                                                            break;
                                                    case CPB_LSBBLOCKNUMBERC :
                                                            CRC =
CrcCalculate(CRC, b2 = GetNextByte(inBuffer,&This->dwInBufferIndex));
                                                            This->dwBlockNumberC
= (This->dwBlockNumberC << 8) + b2;

This->dwFrameLength--;

This->dwDataLength--;

This->dwBlockStart=This->dwOutBufferCount;

This->dwSubFrame=CPB_DATA;
                                                            break;
                                                    case CPB_DATA :
                                                            if
(inBuffer[This->dwInBufferIndex]==CP_FRAME_END) {

This->dwOutBufferCount-=2;
```

```
This->dwSubFrame=CPB_FRAMEEND;
                                                                              break;
                                                                    }
                                                                    CRC =
CrcCalculate(CRC, b2 = GetNextByte(inBuffer,&This->dwInBufferIndex));

outBuffer[This->dwOutBufferCount++]=b2;

This->dwFrameLength--;

This->dwDataLength--;
                                                                    break;
                                                          case CPB_FRAMEEND :
                                                                    gSuccess = TRUE;
                                                                    if
(inBuffer[This->dwInBufferIndex]!=CP_FRAME_END) {
                                                                              AddMessage(
This->fhWnd,"ERROR FIVE");
                                                                              gSuccess =
FALSE;
                                                                    }
                                                                    if
(This->dwBlockNumber==This->dwPreviousBlockNumber) {
                                                                              AddMessage(
This->fhWnd,"Block Reread...");
                                                                    }
                                                                    if
(CRC!=goodCrcValue || !gSuccess) {

This->dwPreviousBlockNumber=This->dwBlockNumber;

sprintf(message,"Block Number:%d
CRC:%d=%d",This->dwBlockNumber,CRC,goodCrcValue);
                                                                              if
```

```
(This->gVerbose) AddMessage( This->fhWnd,message);

This->dwOutBufferCount=This->dwBlockStart;

This->dwInBufferIndex++;

This->dwSubFrame=CPB_NONE;

This->dwFrame=CP_NOFRAME;

DoNack(hComm);
                                                        if
(This->gVerbose) AddMessage( This->fhWnd,"NACK...");

This->dwNackCount++;
                                                        if
(This->dwNackCount==3) fDone=TRUE;
                                                  } else {

This->dwPreviousBlockNumber=This->dwBlockNumber;

sprintf(message,"Block Number:%d
CRC:%d=%d",This->dwBlockNumber,CRC,goodCrcValue);
                                                        if
(This->gVerbose) AddMessage( This->fhWnd,message);

This->dwInBufferIndex++;

This->dwSubFrame=CPB_NONE;

This->dwFrame=CP_NOFRAME;

DoAck(hComm);
                                                        if
```

```
(This->gVerbose) {

AddMessage( This->fhWnd,"ACK...");
                                                } else {

AddMessage( This->fhWnd,AddTick(message));
                                                }
                                           }
                                           break;
                                      }

                                      break;
                              case CP_FINAL :
                                      switch (This->dwSubFrame) {
                                      case CPB_EOT :
                                           CRC =
CrcCalculate(CRC, GetNextByte(inBuffer,&This->dwInBufferIndex));

This->dwSubFrame=CPB_CRC1;
                                           break;
                                      case CPB_CRC1 :
                                           CRC =
CrcCalculate(CRC, GetNextByte(inBuffer,&This->dwInBufferIndex));

This->dwSubFrame=CPB_CRC2;
                                           break;
                                      case CPB_CRC2 :
                                           CRC =
CrcCalculate(CRC, GetNextByte(inBuffer,&This->dwInBufferIndex));

This->dwSubFrame=CPB_FRAMEEND;
                                           break;
                                      case CPB_FRAMEEND :
                                           gSuccess = TRUE;
```

```
                                        if
(inBuffer[This->dwInBufferIndex]!=CP_FRAME_END) {
                                            AddMessage(
This->fhWnd,"ERROR SIX");
                                            gSuccess =
FALSE;
                                        }
                                        else if
(CRC!=goodCrcValue) {
                                            AddMessage(
This->fhWnd,"ERROR SEVEN");
                                            gSuccess =
FALSE;
                                        }
                                        if (!gSuccess) {

DoNack(hComm);
                                            if
(This->gVerbose) AddMessage( This->fhWnd,"NACK...");

This->dwNackCount++;
                                            if
(This->dwNackCount==3) fDone=TRUE;
                                        } else fDone = TRUE;

This->dwInBufferIndex++;

This->dwSubFrame=CPB_NONE;

This->dwFrame=CP_NOFRAME;
                                        break;
                                    }

                                    break;
```

```
                                        }
                                }
                        } while (This->gReading&&!fDone);
                        if (fDone) {

//                                      The following code is used to save the
buffer to a file for analysis
#ifdef  _DEBUG
                                        HANDLE hCommLog;
                                        DWORD dwWrite;


hCommLog=CreateFile("D:\\TEMP\\COMMLOG.BIN",GENERIC_READ | GENERIC_WRITE
,FILE_SHARE_WRITE,NULL,CREATE_ALWAYS,0,NULL);

WriteFile(hCommLog,inBuffer,This->dwInBufferCount,&dwWrite,NULL);
                                        CloseHandle(hCommLog);
#endif

                                        This->gReading=FALSE;
                                        if (This->dwNackCount < 3) {
                                                sprintf(message,"Read %d bytes from
%s",This->dwOutBufferCount,This->gComPort);
                                                if (This->gVerbose) AddMessage(
This->fhWnd, message );         // *****
                                                strcpy((char *) inBuffer,"d=");
                                                i2=2;
                                                for (; i1<This->dwOutBufferCount;
i1++,i2+=2) {
                                                        b2=(unsigned
char)(outBuffer[i1]>>4);


inBuffer[i2]=TranslateDigitHex(b2);
```

```
inBuffer[i2+1]=TranslateDigitHex((unsigned char)(outBuffer[i1]-(b2<<4)));
                                                  }

                                                  err =
PostURL(hWnd,This->gHostName,(unsigned
short)atoi(This->gHostPort),atoi(This->gUID),inBuffer,This->dwOutBufferCount
*2+2,This->gProxyName,(unsigned short)atoi(This->gProxyPort));
                                        }
                                        else      err = 1;
                                        if (err==0) {
                                                  AddMessage( This->fhWnd,"Upload
Successful - please wait...");

NPN_GetURL(This->gInstance,This->gSourceURL,"_current");
                                        } else {
                                                  AddMessage( This->fhWnd,"Upload
Failed");
                                                  Cleanup(This);
                                                  fDone=FALSE;
                                                  CRC=0;
                                        }
                                        /*err = NPN_PostURL( gInstance, gURL, NULL,
dwOutBufferCount*2+2, inBuffer, FALSE);
                                        if( err != NPERR_NO_ERROR ) {
                                                  printf("Error on NPN_PostURL()");
                                        }*/
                              }
                              break;

                    case WM_PAINT: {

                              hdc = BeginPaint( hWnd, &paintStruct );
                    HBRUSH hBr;

                    hBr = CreateSolidBrush(GetSysColor(COLOR_WINDOW));
                    GetClientRect(hWnd,&rect);

                    FillRect(hdc,&rect,hBr);

                    DeleteObject(hBr);

                    for (i = 0; i < gNumLines; i++ ) {
                        TextOut( hdc, 0, (i * 20),
gMessageTextArray[i], strlen(gMessageTextArray[i]) );
                    }

                    EndPaint( hWnd, &paintStruct );
                    break;
                }
                default: {
                    This->fDefaultWindowProc( hWnd, Msg, wParam,
lParam);
                }
            }
            return 0;
}
//================================================
```